NISTIR 6057

CIM Framework Experience Report for 1996

David Flater
Edward Barkmeyer
Evan Wallace
Peter Denno
Mike Iuliano

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899-0001

NIST

NISTIR 6057

CIM Framework Experience Report for 1996

David Flater
Edward Barkmeyer
Evan Wallace
Peter Denno
Mike Iuliano

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899-0001

September 1997

U.S. DEPARTMENT OF COMMERCE
William M. Daley, Secretary

TECHNOLOGY ADMINISTRATION
Gary Bachula, Acting Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Robert E. Hebner, Acting Director

NISTIR 6057

CIM Framework Experience Report for 1996

Contents

List of Figures

This work was funded through the National Advanced Manufacturing Testbed (NAMT) project and the Systems Integration for Manufacturing Applications (SIMA) program.

CIM Framework Experience Report for 1996

David Flater
Edward Barkmeyer
Evan Wallace
Peter Denno
Mike Iuliano

September 18, 1997

Abstract

The National Advanced Manufacturing Testbed (NAMT) Framework prototype is a distributed, object-oriented manufacturing system that serves as a testbed and trial implementation for emerging industry-developed specifications. This report documents the findings of the Framework team in 1996 with respect to version 1.3 of SEMATECH's CIM Framework (CIMF).

1 Introduction

The National Advanced Manufacturing Testbed (NAMT) Framework project[1] has constructed a prototype distributed manufacturing system that serves as a testbed and trial implementation for emerging industry-developed specifications. The first year of the Framework project resulted in a substantial number of problems found with applying version 1.3 of SEMATECH's Computer Integrated Manufacturing (CIM) Application Framework Specification[2] in the context of our testbed architecture. The following sections detail those issues.

Since the version that we reviewed was effectively only a draft, the issues that we describe below should not be interpreted as problems with the CIMF as released; the CIMF has continued to evolve, and our intent was merely to identify potential sources of misunderstanding that could slow the eventual deployment of the specification.

2 Job Management

The purpose of the job control interfaces in CIMF 1.3 was to specify the operations used for creating, starting, and otherwise controlling jobs. However, the intent behind the specific operations chosen was ambiguous. There were two major possibilities, neither of which was fully confirmed or denied by the text:

- The intention was to provide a plug-and-play architecture, where the interfaces were defined with sufficient clarity to enable interoperability between software components from different companies. In this case, the architecture was not clear, and the interfaces defined in the CIMF did not form a coherent functioning whole.
- The intention was to support the union of all architectures, providing enough operations on each interface to enable the usage of any possible model of job control. In this case, the semantics of many of the operations were unclear, some operations were still missing, and the subsets that yielded workable models of job control were not identified.

The following quotes from the Implementation Handbook for version 1.2[1] were later found to support the second theory:

[1] A revision of the Implementation Handbook for version 1.3 has not been released.

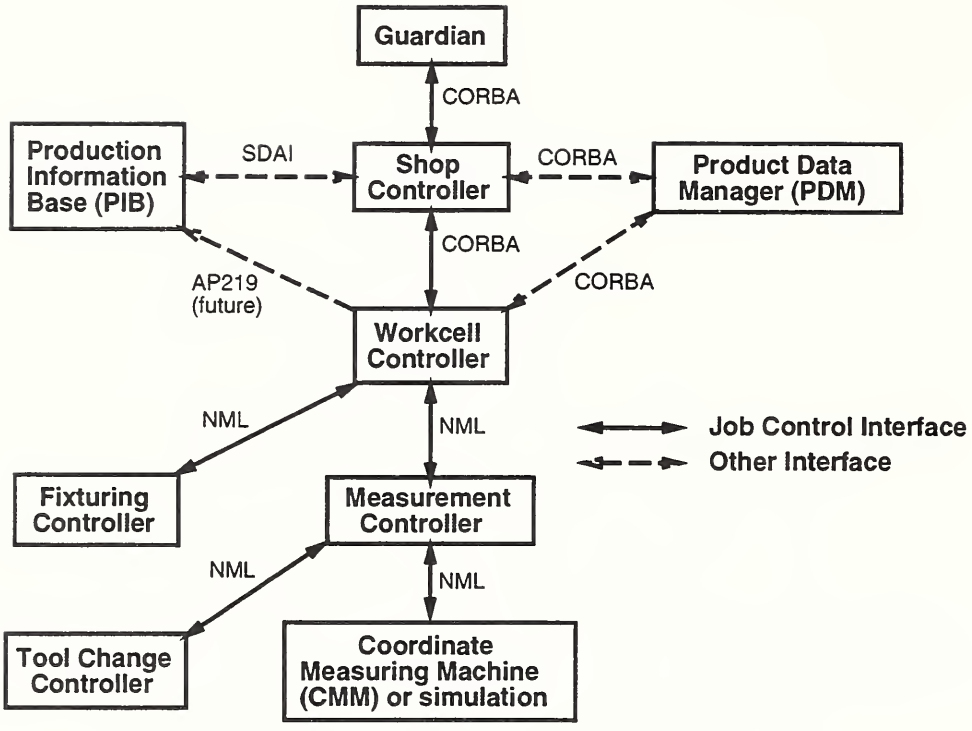

Figure 1: NAMT Framework Architecture

> The SEMATECH CIM Application Framework Specification seeks to unify a wide range of Manufacturing Execution Systems (MES) software by categorizing common features found among them in an object-oriented manner. The specification, however, does not explain how to interpret this common object model to create pluggable applications.
>
> [...]
>
> Interoperability among applications does not automatically result from using CIM Framework objects. In order for two or more applications to be pluggable, they must share the same "binding." A major part of this document is devoted to explaining what SEMATECH has learned about the binding, from low level "nuts and bolts" issues of communication, to placement of CIM Framework object instances across physical machines in a distributed architecture. Even with the same binding, applications do not automatically become "pluggable" via use of the CIM Framework[3].

Unfortunately, the possible "bindings" were not identified, and we were not able to find a binding that would produce a working system within the NAMT architecture (see Figure 1). The NAMT architecture includes several levels of control, including at least the shop level, the workcell level, and the machine level. The components (shown as boxes in Figure 1) are connected by several means of communications (shown as arrows), including CORBA[4]. The shop level includes the scheduling (if any) as well as the dispatching of tasks to workcells; the workcell level *may* correspond to the CIMF Machine object, and the machine level (exemplified in Figure 1 with a Fixturing Controller and Measurement Controller) *may* correspond to the CIMF Machine Resource object. However, it really is not clear which classes from the CIMF should be used by the controller at each level, and any given mapping that we attempted to make between the CIMF and the NAMT architecture posed serious problems.

The NAMT workcell is an abstraction comprising one or more mechanical subsystems (i.e., machines) and possibly an operator. The purpose of the abstraction is to make a meaningful division between shop-level control and the lower levels of control. While the shop controller manages processes at the routing level (which tasks will be done at which stations), the workcell controller manages all processes at one station, a level between the shop and the machine. An example of

a workcell task would be: "Rough cut workpiece xxx." The workcell controller decomposes this into steps performed by its components: "(1) Get the operator to load workpiece xxx. (2) Get the machine to run NC-program yyy. (3) Do the following extra things. (4) Finish." Note that the NC-program defines a complex task at the next level down, which we call the machine level. And for the operator (or a positioning robot) there is a set of instructions for loading and positioning the part that is analogous to the NC program for the cutting machine.

The CIMF contains multiple classes that seem to map imperfectly to the workcell. One of the causes of this is the necessity of distinguishing the persistent view of a controller (any controller) from the non-persistent view. The question "Are you turned on?" cannot be addressed to the controller itself, since it cannot answer if it is turned off. Instead, a persistent database component such as the Production Information Base must serve as its proxy to answer questions relating to power status, scheduled downtime, and so on. Unfortunately, even this understanding does not suffice to explain the classes in the CIMF, because control operations like pause, resume, and makeActive that only make sense for the non-persistent, live controller are included on classes that can only pertain to the persistent view. The flow of control that results is unusual.

The other side of the problem is that the CIMF contains *no* classes that map to the shop controller. ProcessJob and ProcessJobManager are both specialized to the workcell level, and ManagedJob and JobManager are intended simply as abstract superclasses. Furthermore, neither JobManager nor ProcessJobManager has an associated set of states. There is a distinction between job states and job manager states that must be preserved – the action of pausing the dispatcher, allowing currently executing jobs to complete normally but not initiating any new activity, is distinct from the action of pausing all jobs that are currently running. The most appropriate set of states for both the shop and workcell controllers (job managers) seems to be that of the MachineResource class, but the rest of the MachineResource class is not applicable above the machine level. According to the comments that we have received from SEMATECH, the MachineResource class is slated to be merged with Machine. Also, like ManagedJob, MachineResource lacks "fault" states. It is not clear how a controller can inform its superior and/or its operator that human intervention is needed, and there is no state flag that can be set to indicate that a job or a controller is stuck waiting for help.

Ultimately, we found no consistent interpretation of these CIMF classes that allowed us to map them to components in our system. It was always necessary to take pieces of one class and pieces of another. In some cases, there were superfluous operations; in others, necessary operations were missing.

In the process of trying to find a consistent interpretation of the CIMF, we identified four possible models of job control – four different generic approaches for choreographing the component interactions that would enable our system to function correctly. That which distinguishes one model of job control from another is the allocation of objects among physical machines. These models would correspond to different CIMF "bindings" if the CIMF provided the classes needed for the shop controller and defined all of the communication channels between the shop and workcells. This is the subject of a separate paper[5].

The summary comments that were delivered to SEMATECH can be found in Appendix B.

3 Specification Management

The Document Management Component of the CIMF is incomplete as a specification for an interface to a commercial Product Data Manager (PDM) or Electronic Document Manager (EDM), even for the limited needs of a production system. The Document Management Component contains the classes (and concepts) DocumentManager, DocumentSpecification, and DocumentRevision and the misnamed Version Manager, which actually deals with Engineering Change Orders. The following additional concepts are needed:

1. DocumentType. In addition to its "name," which identifies a particular document instance, such as a particular NC-Program, it is also important to model its "type," which identifies the kind of document it is, e.g., that it is an NC-Program and not a CAD model. In a commercial PDM/EDM, the behavior of a Document – workflow, access rules, versioning rules,

etc. – is associated with its type. (It is, of course, possible to embed the type in the naming convention, but this is often inconvenient to both the sender and the receiver of document access operations.)

2. DocumentForm. A given revision of a given specification may need to be represented in different forms (or file formats) for different consumers, e.g., a drawing may be stored in the native format of the CAD package in which it was created, but also in a standard display form for use in operator displays. In general, a software package retrieving the document must be able to specify the form it wants.

3. DocumentRelationship. For engineering purposes, it is important to model and maintain certain relationships among documents and other "business objects" managed by the PDM/EDM: "contains" allows for the management of objects that are packages of documents, such as the collection used at a given workcell; "depends on" captures which versions of other documents a given document depends on, so that ripple effects of changes can be determined; "derived from" supports traceability of specifications; "supersedes" supports version management. This capability, however, does not seem to be needed for production-only usage of the PDM/EDM.

In addition, the concept of version management is critical to automating the transmission of engineering specifications to production. In PDMs, the following concepts are all aspects of version management.

- "version" – a specification modified for a particular use,
- "revision" – a change in the specification,
- "effectivity" – the range of dates or product instances to which a particular revision applies, and
- "change notice" – a directive to make a set of revisions simultaneously effective in production.

The CIMF supports a much simpler model of these concepts. The CIMF model seems to be "linear effectivity by date," that is, the "current" revision of a document is in effect until the date on which the "next" revision becomes active. Presumably, making related specifications simultaneously effective is the responsibility of the user. One would expect this to be the function of the ChangeNotice, but no relationship between the ChangeNotice and any specification revisions is modelled or even discussed. Moreover, the ChangeNotice does not itself have an effective date! Apparently its "activation" makes it effective as of that moment.

The CIMF Version Management model may be adequate for semiconductor fabrication, but it is inadequate for the manufacture of electro-mechanical parts. Because of staggered deliveries of materials and substitutes, and because of pre-emption of resources by higher-priority jobs, it is common for multiple batches of the same product to be on the shop floor in different stages of processing, and each batch may have a different set of processing specifications. This makes it possible for multiple revisions (or versions) of a specification to be "active" at any given time, with the effectivity tied to batch or lot numbers, or to the date on which the lot "started." A Change Notice itself must have an "effectivity," by date or batch/lot and possibly by location (i.e., which factories), and that effectivity is usually set some time in advance of its occurrence. And, it is the Change Notice effectivity that defines the effectivity of the related production specifications for the target facility. Supporting this capability requires a number of significant changes to the CIMF model.

Finally, the CIMF model of "document content" is "any." This is too general, as it does not make clear whether the user retrieving a document content will get a file *name* or a file *text*. While we agree that there is a need for other possibilities as well, it is necessary to make clear whether an operation gets the file text, and how a large file text can be handled. (Appealing to other CORBA services is a reasonable solution, but it needs to be documented.)

In addition to these major issues, there are also many apparent anomalies, unexplained changes and outright errors in the comments and descriptions for the specification management interface. A detailed enumeration of these can be found in Appendix C.

4 Machine Management

The major problem with the Machine Management component is that several classes combine concepts that must be physically separated in useful implementations. With respect to factory resources, the agent who manages *description* of a resource (the persistent view of a resource, as discussed above), and the agent who manages *control* of a resource (the non-persistent view) should be distinct. But in the CIMF, they are not. Examples:

- The class Machine has control operations like pause and resume, but also descriptive operations: configuration operations like addProcessResource, and scheduler "gating" operations like reserveFor.

- The class MachineResource, and all its subtypes except for ProcessResource, have purely descriptive operations, but MachineResource is modelled as a subtype of MovementResource, which accepts "handoff commands," i.e., transfer control operations. And the ProcessResource subtype of MachineResource also has the control operations doProcessJob and canMakeActive.

- The description of the ProcessJobManager class (the obvious model of the controller) indicates that it should *receive* new Job invocations from the ProcessResource and new material flow invocations from the TransferResource.

The CIMF apparently models an agent who maintains the persistent information and also serves as a front-end for the actual machine controllers, using other standard and non-standard protocols for the actual communications. This is certainly a possible architecture, but merging the control and description operations into one class forces it to be the only possible architecture. And this architecture is *not* desirable for the future – it makes it impossible for future controllers to conform to the CIMF as a standard by providing the standard control operations themselves, and thus eliminate the front-end. By comparison, separating the control and descriptive operations into different classes allows many architectures – one implemented agent can support one class or multiple classes.

A lesser, but also important, problem is the handling of "machine setup." A "process capability" is the ability of a given machine to perform a given process with some configuration; a "machine setup" is a specific machine configuration and/or the work needed to achieve that configuration. In the general case, the relationship between the two is many-to-many: a single setup may support multiple processes, and a single process may be possible with different setups of a machine or require different setups on different machines. It seems that the CIMF model is intended to hide the machine setup altogether, so that a machine at any given time has a "set of ProcessCapabilities," i.e., those that correspond to its setup. The weakness of this model lies in the fact that machine setup is also a process that consumes resources and time, and good schedulers need to know *both* what process capabilities are currently present on a machine – what the setup *state* is – *and* what resources need to be scheduled to create a particular set of capabilities on otherwise idle machine – what the setup *process* requires.

Finally, the aspects of machine description and control that relate to material storage and movement are inadequately explained and in some cases very confusing. The problem centers on the terms Port and Portal, whose definitions can be easily construed to overlap and whose operations are conflicting. It appears that the meaning of some of these objects and relationships changed from a prior version and not all of the operations and terminology were made consistent between the "materials transfer" models and the "materials processing" models. The problem is exacerbated by the attempt of the CIMF to keep pace with the terminology and material handling models of a separate ongoing semiconductor industry standard. There should be a unifying model here, but it is not present in CIMF 1.3. This makes it difficult to develop a clear appraisal of the utility of this model to the analogous objects and operations in the manufacture of electro-mechanical products.

A detailed enumeration of individual problems can be found in Appendix D.

5 Materials Management

The Materials Management component is the one component of the CIMF that is likely to be directly useable in the manufacture of electro-mechanical products. The one complaint is that many of the operations assigned to subtype SCMaterialManager (semi-conductor materials) should properly be assigned to the general (super)class MaterialManager, as they relate to Lot management in general, and not to wafer management in particular.

For electro-mechanical part manufacture, however, the model of Durables (Tooling and Fixtures) is not adequate. This is to be expected, because tooling for semiconductor fabrication is much simpler. Unfortunately, (reversing the behavior of SCMaterial versus Material) several over-simplifications have been introduced into the general class Durable. These must be modified to produce a generally useful model.

A minor point is that the nomenclature "Product" is used in the CIMF (and in semiconductor fab jargon) to mean "Workpiece," but this terminology does not generalize to discrete parts, or even to the discrete aspects of semiconductor manufacture (dice, mount, test and package). In almost all discrete parts industries, the term "product" refers to a *type* of manufactured object, as opposed to "instances in process," which are called "parts," "pieces" or "units." (The term is used for the in-process material in process industries, but there is no equivalent "piece" aspect.)

A detailed enumeration of individual problems can be found in Appendix E.

6 Conclusion

As a result of the work done by the Framework team, ambiguities and technical problems that threatened the applicability of the CIM Framework were reported to SEMATECH while the document was still in its early stages. While some problems in machine management and document management had been discovered by other reviewers, our analyses of job management and machine setups provided unique input.

SEMATECH intends for version 2.0 of the CIM Framework to be a complete and polished specification, the end result of the 1.X revision cycles. We are gratified to have contributed to that process.

References

[1] Howard M. Bloom and Neil Christopher. A framework for distributed and virtual discrete part manufacturing. In *Proceedings of the CALS EXPO '96*, Long Beach, CA, October 1996.

[2] Lawrence Eng, Ken Freed, Jim Hollister, Carla Jobe, Paul McGuire, Alan Moser, Vinayak Parikh, Margaret Pratt, Fred Waskiewicz, and Frank Yeager. *Computer Integrated Manufacturing (CIM) Application Framework Specification 1.3.* SEMATECH, 2706 Montopolis Drive, Austin, TX 78741 U.S.A., 1996.

[3] Ken Freed. *Implementation Handbook for the Computer Integrated Manufacturing (CIM) Application Framework Specification 1.2*, page 1. SEMATECH, 2706 Montopolis Drive, Austin, TX 78741 U.S.A., 1995.

[4] OMG home page. <URL:http://www.omg.org/>.

[5] David Flater, Edward Barkmeyer, and Evan Wallace. *Four Models of Job Control.* National Institute of Standards and Technology (forthcoming), 1997. To be available from the National Technical Information Service, Springfield, VA 22161 U.S.A.

A Comments on CIMF "Abstract" Interfaces

What is the purpose of OwnedEntity?

The class OwnedEntity is a pure factorization ("mix-in") that seems to be some kind of implementation convenience. Its only attribute is `attribute any owner;`. Such a model is a guideline, not a class: an OwnedEntity is something that has an "owner" attribute. But there is no commonality of the type of owners. This characteristic is clearly better modelled by giving each "OwnedEntity" type its own owner attribute with the proper object type. A few OwnedEntity types might have to settle for "object," because different members of the class can be "owned" by objects of different classes. But at least this would encourage the documentation to specify the range of classes intended in that case. In any case, "any" is too broad – all owners must be objects. Of course, the implementation conscious might want to use "any," so that the owner could be implemented as a "string," but in this case, "string" is a just a poor model for NamedEntity.

Thus the class OwnedEntity should be discarded and each of its current subtypes should declare the owner attribute properly. If there is an implementation intent here, then it should be explained and the use of OwnedEntity as a supertype should match that intent.

Redundant operations on ComponentManager

What is the relationship between the operations ComponentManager::makeStartingUp() and Resource::startUp(), ComponentManager::makeShuttingDown() and Resource::shutdownNormal(), ComponentManager::makeStopped() and Resource::shutdownImmediate()?

Because ComponentManager is modelled to inherit from Resource, it will actually have all 6 operations. It seems that either the operations on Resource are intended to be "virtual" and should be removed from the IDL, or the operations on ComponentManager are redundant and should be removed from the IDL. In any case, there should not be two sets of operations and the spelling should be made consistent.

B Detailed Comments on CIMF Job Management Interfaces

No Class for Shop-Level Job Manager

The ProcessJobManager class corresponds to what we would call the workcell level of control. It seems to us that most factories will have at least two levels of control: the workcell level, and the shop level. The CIMF provides no class for the shop-level controller.

We believe that the control interface at the shop level and at the workcell level should be nearly identical; they should each inherit the same interface from a generic superclass, i.e., JobManager. This then extends easily to hierarchical control with more than two levels.

No States for Job Managers

Neither JobManager nor ProcessJobManager has an associated set of states. There is a distinction between job states and job manager states that must be preserved. Only having one or the other is not adequate; pausing all managed jobs is different than pausing the controller. The most appropriate set of states for both the shop and workcell controllers (job managers) seems to be that of the MachineResource class. But the rest of the MachineResource class is not applicable above the machine level, and according to the comments we have received, the MachineResource class is slated to be merged with Machine.

Superfluous and Misleading Job Control Operations

Feedback that we have received previously from SEMATECH has done much to clarify the intended job control architecture of the CIMF. In a nutshell, we now understand the intended job control architecture to be the following:

1. The supervisory controller invokes createJob in the subordinate to create a job.
2. Operations of the form make-something (makePausing, makeNotPaused, makeAllManagedJobsAborting, etc.) are the commands to control the state of jobs and/or job managers.
3. The status of jobs is reported back to the supervisor through events.

However, this interpretation leaves us with the following issues:

1. The doProcessJob operation is superfluous. Instead you should use makeQueued and/or makeActive.
2. The informCompletedJob operation is superfluous. There is a JobCompletedEvent.

Based on our own experiences, we would be inclined to add operations along the lines of informJobPausing, informJobAborting, informJobAborted, and so on, and to stop using events for closing the control loop between supervisor and subordinate. However, this is not consistent with the direction that the CIMF has chosen.

No Fault Handling

In real manufacturing facilities, being able to deal intelligently with unexpected failures is among the most important design considerations. Unfortunately, this capability does not appear to have been provided in the CIMF.

Let us consider the scenario where an operator has instructed the controller to pause a particular job, but while the job is approaching the next convenient pause point, a piece of machinery gets stuck and somebody needs to go give it a nudge. When we try to communicate this fact using the CIMF facilities, we find two problems.

First, we find that the states associated with ManagedJob do not adequately distinguish between normal states and fault states. As far as we know, our unfortunate job remains in the concurrent state described by Executing, Pausing, NotStopping, NotAborting. We need an additional qualifier to indicate that the job requires human attention. Simply throwing the job into the Aborted state is unacceptable, since this will needlessly ruin a workpiece and may even induce higher-level jobs to abort similarly.

Second, we find that it is not clear how a controller can inform its superior and/or its guardian (operator interface) that human intervention is needed. An operation or event is needed for this purpose.

Missing Operations

JobManager has allQueuedManagedJobs, allActiveManagedJobs, allFinishedManagedJobs, allStoppedManagedJobs, and allAbortedManagedJobs. It lacks allCompletedManagedJobs and allCancelledManagedJobs.

C Detailed Comments on CIMF Specification Management Interfaces

P. 161, clause 4.2.4.1: "document specifications" should be "specification documents."

P. 161, Document Management component: There are 3 missing objects in the OMT diagram:

a. DocumentType. That is, a "type" which specifies the behavior, workflow, access rules, allowable states signoff rules, allowable forms, etc. for a DocumentSpecification. The concept DocumentType is missing from the CIM Framework, but is critical to all WorkFlow Management software systems.

b. DocumentForm. The user needs to be able to represent multiple forms of the same conceptual revision. A revision has a master form (the original), e.g. Pro/E native form, and derived forms, e.g. IGES, DXF. Each form has its own documentContent. So there seems to be a missing object here. That is, a DocumentRevision has DocumentForms and each DocumentForm has a formType (string), which might be its (NamedEntity::)name, and a documentContent.

c. DocumentRelationship. Documents may have relationships important to configuration management. Specific subclasses, e.g. ProductSpecification, may have specific modelled relationships, but the PDM must be aware of "dependency" relationships. In addition, users may define other relationships, such as product families. A document relationship has a one-to-many relationship with DocumentRevisions (not Specifications, we think) and MAY have a 1-to-1 relationship with a particular DocumentRevision (sometimes parent – relationship based_on, sometimes dependent – relationship depends_on).

P. 162, DocumentManager::createDocumentNamed: The new DocumentSpecification should be related to a "DocumentType" by a parameter in the create operation. See above.

P. 163 Class DocumentManager: The operation removeDocumentNamed has a misspelling of the exception name: DocumentSpecificationHasRevisionsNoRemoveSignal.

P. 163 Class DocumentManager: The operation sequence<DocumentRevision> allRevisions(); is not an operation on the DocumentManager object; it should be sequence<DocumentSpecification> allDocuments();

P. 163 Class DocumentManager: findDocumentNamed should raise DocumentSpecification-NotFound if it fails, rather than DocumentSpecificationRetrievalFailed. It is not clear why the v1.2 specification was changed in this regard. ...NotFound indicates there is no specification with the given name, which is the case here. NotFound is neither more nor less of a problem for "find" than it is for "remove," so why the difference?

P. 164 class DocumentRevision: Only one "active" version is an extremely strong limitation. The assumption being made is that the "effectivity" of one revision [first-date of use, last-date of use] does not overlap the effectivity of any other. In the general case, particularly of complex assemblies, this is not true – one revision may describe parts currently in the first fabrication phase, while another describes parts in a late fabrication phase, and a third describes parts in some post-fabrication assembly phase, all of which are currently active because of feedback, tooling changes, materials changes, etc.

P. 164 DocumentRevision attribute documentContents: Since the type is "any," how does the user determine what the "form" of the contents is?

P. 164 class DocumentRevision: Is holds() the same operation as _get_documentContents(), i.e. the retrieval operation on the documentContents attribute? If not, what is it?

P. 164 class DocumentRevision: It appears that holds(), or _get_documentContents, could fail and be able to raise DocumentContentsRetrievalFailed, an exception that is not documented. When the DocumentManager classes are implemented in a PDM, the contents files may or may not be resident within the PDM itself, and thus their retrieval can fail even when the Revision object is "intact."

P. 165 class DocumentRevision: The user needs to be able to represent multiple forms of the same conceptual revision. See DocumentForm above.

P. 170 Class DocumentSpecification: What is meant by "lineage order" for the sequence of DocumentRevisions? Note that addRevision and createRevisionNamed do not provide for positioning, so "lineage order" is derived from some other datum or behavior: activationDate? chronological order of add/create invocations?

P. 170 Class DocumentSpecification: Operation findRevisionNamed should raise DocumentRevisionNotFound if it fails, rather than DocumentRevisionRetrievalFailed. It is not clear why the v1.2 specification was changed in this regard. (See above argument.)

P. 170 Class DocumentSpecification: For operations addRevision and createRevisionNamed, the descriptive text does not make clear the functions of these operations.

Is addRevision intended to be a "copy constructor," i.e. an operation that makes a new DocumentRevision object with the same (initial) contents as an existing one? If so, the text should say that.

It seems that createRevisionNamed_with also "adds a document revision to the list of revisions that are associated with the DocumentSpecification." Otherwise why would it be an operation on a DocumentSpecification object?

The alternative interpretation is that one uses createRevisionNamed_with to construct a DocumentRevision object and addRevision to link it to a DocumentSpecification, but then addRevision should return void and createRevisionNamed_with should not be an operation on DocumentSpecification.

P. 170 DocumentSpecification::createRevisionNamed_with: This operation seems to have a missing parameter: Form of the versionedObject – AP203file, ASCIItextfile, DXFfile, etc.

P. 170 DocumentSpecification::createRevisionNamed_with: If versionedObject is a BIG file, how does it get moved? Is the client supposed to have it in a large "string" in memory? Or is moving the file a separate service supplied by the client's system and used by the DocumentManager?

We suggest that both cases should be supported. Filename should be one of the types permitted for versionedObject and the DocumentType for the DocumentSpecification should determine whether the file is copied or pointed to. This means that there is really another operation:

```
/* creates a revision from a file accessible to the DocumentSpecification,
 * possibly by invocation of some remote file access services.
 * Whether the file is copied into the Specification or simply pointed
 * to (and retrieved on a request to retrieve documentContents) is an
 * implementation and policy issue.  */
DocumentRevision createRevisionNamed_fromFile (
    in string revisionName, in string documentFormat,
    in string filename)
    raises (DocumentRevisionDuplicateSignal,
            DocumentTextRetrievalFailedSignal);
```

P. 170 class DocumentSpecification, removeRevisionNamed: The comment that describes the operation should read: "Remove the named DocumentRevision associated with the DocumentSpecification."

P. 174 class ChangeNotice:

The parameters to operation createChangeNoticeNamed_inDocument don't match the OMT model on p. 161. The OMT model allows a (production) ChangeNotice to refer to one or more documents, but the operation, and the attribute ChangeNotice::specification (p. 174) only allow ONE DocumentSpecification to be associated. The relationship modelled in the IDL is that the ChangeNotice itself is a document, that has text, is subject to revision, etc. This relationship is between the ChangeNotice object and the corresponding DocumentSpecification and this is one-to-one. So the OMT model should be corrected.

P. 172 classes VersionManager and ChangeNotice:

A production "change notice" is itself a document that references other documents – a feature noticeably lacking from either DocumentRevision or ChangeNotice. (This could be addressed by

DocumentRelationship, as indicated above.) In our experience, a change notice going to the floor requires a separate approval process from the approval process for the documents it references. So these two ideas should be separated. Any document can incorporate other documents by reference and be subject to some approval path. Approvals have signatories and dates. Change notices are a kind of document that has in particular an effectivity date (that is separate from the approval dates). Note also that effectivity of the ChangeNotice is what really determines whether and when a Document is "active."

Both the OMT model and the IDL should be corrected to support this understanding. ChangeNotice should have two additional attributes:

```
/* set/get the set of DocumentRevisions made effective by this
 * ChangeNotice. */
attribute sequence<DocumentRevision> associatedRevisions;

/* set/get the date on which all associatedRevisions become effective
 * in production. */
attribute TimeStamp effective_date;
```

P. 174 class ChangeNotice: makePreparingToActivate does not "Approve a change notice." Rather it submits the draft change notice to the approval officer for signature, as indicated in the state table (p. 176). makeActivated should be "approve and activate a change notice."

P. 174 class ChangeNotice: "default SignOff set" is a curious term. The real *default* Signoff list would be automatically attached to all ChangeNotices, but that is *not* what these operations modify. It appears that the notion is misnamed. What is being modified by these operations is not the Default Signoff List, but rather the actual Signoff List for *this* ChangeNotice. Note also that the ordering of the Signoff List may be important (i.e. a routing), as the comment under defaultSignOffSet() says. So the semantics is not usually a Set. Change the operations to: addNameToSignOffList (...); and removeNameFromSignOffList (...); and signOffList (...);

P. 172ff, classes VersionManager and ChangeNotice:

This model of signoff creates more problems than it solves. If multiple signoffs are required to get from PreparingToActivate to Activated, there are intermediate states that don't have names, and a date and some form of signature, possibly electronic, that needs to be associated with the Sign Off operation. Our general concern is that the CIMF is flirting with Workflow Management here, but not producing a robust model, and thus saddling subtypes with a dubious half-model. Recommendation: "drink deep or taste not." Dump the DefaultSignOffSet operations.

D Detailed Comments on CIMF Machine Management Interfaces

Persistent and non-persistent views of the Machine

It is necessary in any implementation of the CIM Framework to distinguish between the persistent view of the Machine and the non-persistent "dynamic" view.

- The persistent view describes the past, present and future states of the hardware – configuration, staffing, maintenance schedule, etc. The "persistent" view is so-called because the objects in it must be available (to planning systems) even when the machine and its controller are physically shut down. We foresee the persistent view being supported by a database-like equipment management agent.

- The "dynamic" view describes the current operating states, the current ProcessJobs and TransferJobs, and other transient information one would expect to be supported by the machine controller. And we foresee the object server supporting the dynamic view being the machine controller.

The controller can expect to have access to the persistent information provided by the equipment manager, and therefore the "controller" server can respond to messages that require persistent information, if this is considered convenient. But the equipment manager can *not* expect to have access to the information possessed by the controller.

From the proposed revisions to CIMF 1.4, we conclude that the class Machine, together with its MachineResources, is intended to represent the persistent view, and the MachineManagement component is intended to represent the equipment management server. From CIMF 1.3, we also concluded that the ProcessJobManager and TransferJobManager classes represent the "dynamic" or "controller" services. (This dichotomy helps to explain the existence of multiple classes in the CIMF that seem to map imperfectly to the same manufacturing object – the "machine.")

If this is the case, CIMF 1.4 should NOT contain in the MachineManagement component ANY operations which are to be understood as commands to the system to perform a physical function! All such operations are properly directed to the xxxJobManager classes. Related operations on the MachineManagement classes *record* the logical and physical changes of state of the hardware systems. (In many cases, the clients for these operations will be the controllers.) This clarification should appear in the comments describing those operations in CIMF 1.4, and in a few cases, the operations should probably be renamed to avoid confusion.

Recommendation:

a. Delete the following operations on class Machine: pause, resume, canMakeActive, doProcessJob, informProcessJobStarted, informProcessJobCompleted, informTransferJobStarted, informTransferJobCompleted, makeAllProcessJobsAborting. transportResourceAvailable and the four related "events."

b. Delete the following operations on ProcessResource: doProcessJob, canMakeActive, makeActive

c. On MachineResource remove the comment "Perform any activities associated with ..." from the operations: materialLocation_received, materialLocation_sent

d. On PortMaterialLocation (to be MaterialPort?) remove the comment "Do any activities based on this." from the operations: material_hereAt, material_goneAt

(We recognize that this particular CHOICE of assignments of server functionalities to interface classes is controversial. We would be satisfied with any resolution which makes a clear distinction. Exactly which classes do we propose to be provided by the vendor of a machine controller?)

Capabilities vs. Setups

"Capability" seems to have two different meanings as used in the ProcessResource operations. In the definition of class ProcessCapability, and in ProcessResource::possibleCapabilities(), it means a collection of processes the resource can perform:

```
/* Answer a sequence of ProcessCapabilities representing the total set
 * of designed processes for this ProcessResource. */
ProcessCapabilitySequence possibleCapabilities();
```

But in currentCapability() and assignedCapabilities(), it apparently means "Setup," viz.:

```
/* Set and get the single ProcessCapability representing the current setup
 * (configuration of consumables and fixtures) of this ProcessResource.
 * This ProcessCapability must be in the assigned list. */
attribute ProcessCapability currentCapability;

/* Answer a sequence of ProcessCapabilities representing setups allowed
 * for this ProcessResource. This must be a subset of the total possible
 * capabilities. */
ProcessCapabilitySequence assignedCapabilities();
```

If the definition of class ProcessCapability is right, then currentCapability() and assignedCapabilities are either misnamed or misused.

These notions are *not the same*, and in general there is a many-to-many relationship between "process capabilities" and "machine setups." That is,

- for many machines, a single standard setup may support more than one "process," depending on how finely the concept "process capability" is delineated. (1 setup, multiple process capabilities)
- for some machines, different standard setups of the machine may support some common processing capabilities, and the "setup" for the same "process" may be different for an alternative machine. (1 process, multiple possible setups)

Furthermore, a machine Setup is an activity in its own right – it takes time and staff, and for many machines it may require material deliveries. Thus it is a non-process activity that can be "scheduled." Setups should have a name, so that schedulers can determine eligible machines for a process and/or cost of re-setup. The concept "machine setup" is generic to many dissimilar manufacturing industries.

Note that, at least in mechanical parts manufacture, there is a distinction between Machine setup and Part/Product setup. Machine setup refers to the configuration of the machine for one or more "process runs," while Part setup refers to the (usually manual) operations of loading and positioning individual workpieces (Products) for processing. Machine setup should be scheduled in, while Part setup is just factored into the "Operator instructions" and time for a specific process. And the parameters/settings for the machine Setup and the Part processing may be different. This distinction is correctly supported in CIMF 1.3 by setupSettings for Machine and currentProcessSettings for ProcessResource.

Recommendation:

a. Introduce a new "object class" – MachineSetup or ResourceSetup – to represent the Setup concept, and copy setupSettings from Machine into this class. (Leave ProcessCapability as is.)

```
/* Machine Setup models the configuration of a machine for one or
   more "process runs".  A given setup may be specific to a single process,
   or "standard" in the sense that it supports a common set of processes.
   The configuration defined by a MachineSetup is supported by
   specifications for the process of configuring the machine.
 */
interface MachineSetup : NamedEntity {

  /* Answer the set of ProcessCapabilities supported by the Machine with
   * this MachineSetup. */
  ProcessCapabilitySequence capabilitiesSupported();
```

```
    /* Answer the specification for performing the reconfiguration. */
    ProcessDefinition definition();

    /* Return the setup-specific settings for the Machine with this Setup. */
    sequence <setting> setupSettings();

}
```

b. Change ProcessResource::currentCapability to currentSetup:

```
    /* Set and get the single MachineSetup representing the current setup
     * (configuration of consumables and fixtures) of this ProcessResource.
     * This MachineSetup must be in the assigned list. */
    attribute MachineSetup currentSetup;
```

c. Change ProcessResource::assignedCapabilities to assignedSetups:

```
    /* Answer a sequence of MachineSetups representing setups allowed for
     * this ProcessResource. */
    sequence <MachineSetup> assignedSetups();
```

Machine::setupSettings

```
/* Set and get current setup-specific settings for the Machine. */
attribute settingSequence setupSettings;
```

Are these settings for the Machine (generally) or for some particular ProcessResource? The CIMF distinguishes between a Machine and its ProcessResources and thus it appears that setupSettings is an attribute of the wrong object. Or is there a hidden assumption that a Machine has at most one ProcessResource?

In any case, the "setupSettings," as distinct from the current process settings (which are correctly supported by ProcessResource), are part of the machine setup. If the above recommendation is taken, then this should be an attribute of MachineSetup.

Machine::processCapabilities

```
/* Answer ProcessCapabilities this machine has. */
ProcessCapabilitySequence processCapabilities();
```

Which "capabilities?" The ProcessResource distinguishes "assignedCapabilities," "possibleCapabilities" and "currentCapability." It appears that the intention here is really ProcessCapabilities and that the intention is "currentCapabilities," as distinct from that attribute of the ProcessResource which is called "currentCapability," but means the (single) "current Machine Setup." This just needs to be clarified in the comments.

Machine::reserveFor

```
/* Reserve a Material for production. Material may be reserved for
 * only one entity. Return true if successful. */
boolean reserveFor (in NamedEntity requester);
```

```
/* Unreserve a Material for production. */
void unreserve();

/* Return true if the Material has been reserved for production. */
boolean isReserved();

/* Return the NamedEntity for which the Material has been reserved. */
NamedEntity reservedFor();
```

The word "Material" should be "Machine" in all occurrences here. These are operations on the Machine, not on some Material in it.

Also, machine reservation is not just for "production." Rather a machine is reserved for some particular upcoming production or setup or maintenance task. NamedEntity is obviously a synonym for "any" here, and far too gross. A Machine is really reserved *by* some Person (with authority) *for* some not necessarily identifiable (because it may be future or engineering) Job.

Recommendation:

a. Change reserveFor to:

```
boolean reserve (in Person requester, in string purpose);
```

When it is reserved for a Job, the "purpose" can be the Job identifier.

b. Change reservedFor to:

```
/* Returns the Person who has reserved the Machine, or NIL. */
Person reservedBy();

/* Returns the purpose for which the Machine is reserved, or NIL. */
string reservedFor();
```

E Detailed Comments on CIMF MachineResource Component

Machine::findMaterialNamed

```
/* Answer the material with this name or nothing. */
Material findMaterialNamed(in string materialName);
```

This operation has the form of the name-lookup operation on MaterialManager. Whatever this operation is supposed to mean, it should not be so named.

Why is this an operation on Machine? Is this supposed to find the Material if this Machine has it? If so, this operation is conceptually:

```
/* Answer true if the Material is somewhere on this Machine. */
boolean findMaterial(in Material aMaterial);
```

or perhaps better:

```
/* Answer the location of the Material in this Machine, if it is
 * present, or NIL if it is not present on this Machine. */
MaterialLocation findMaterial(in Material aMaterial);
```

Machine: MovementResource vs. MachineResource: MovementResource

This is an aspect of a general architectural problem with the Framework. The implementation is confused with the service interfaces.

If an implementation provides instances of *both* Machine and MachineResource, then the transport protocols defined for MovementResource should *not* be defined for the Machine class, and Machine should *not* inherit from MovementResource. And in practice, the transfer protocol operations should be directed to the MachineResources that inherit from MovementResource.

It is envisioned that a *single* CORBA "server" may support an instance of Machine *and* instances of all its MachineResources. So the same server implementation will in fact provide all elements of the "MovementResource" protocols. The Framework will support this architecture *without* overloading Machine with the operations that are properly assigned to the other classes supported by the same server.

If an implementation may elect *not* to provide instances of MachineResource, then the Framework model must be *altered* to support that implementation. That is, that implementation is supporting a class which inherits from both Machine and MachineResource and calling it "Machine." This behavior should not be specifically included in the CIMF.

Recommendation: Change Machine so that it does *not* inherit from MovementResource.

What is a PortResource?

Does a PortResource model a *single* "port?" The following operations apparently make this assumption:

```
Machine::firstAvailablePort
PortResource::makeInputPort, makeOutputPort, makeInputOutputPort
PortResource::isLoaded
```

That is, these operations seem to presume that the PortResource has only one externally accessible container location (PortMaterialLocation). E.g. firstAvailablePort returns a PortResource and not a MaterialLocation, so the PortResource is either totally available or totally unavailable. And makeInputPort makes ALL container receptacles at the port incoming. And the definition of isLoaded refers to *the* PortMaterialLocation.

But the IDL (and the OMT diagram) indicate that a PortResource can have more than one PortMaterialLocation, i.e. externally accessible container location. If multiple container locations are accessible, then one would expect that the individual locations would be set to input or output and be separately available or occupied. Something is wrong here.

PortResource::isLoaded

```
/* Check to see if any Material is at the PortMaterialLocation (i.e.,
 * a PositionalContainer/Cassette). */
boolean isLoaded();
```

In general, this operation is on the wrong object. It should be an operation on PortMaterialLocation.

Does this operation mean "check to see if there is a PositionalContainer in *any* PortMaterialLocation at the port?" If so, the comment should be restated, as it clearly assumes that there is only one PortMaterialLocation.

If the assumption that there is only one PortMaterialLocation is valid, this service is redundant, as the service is already provided by MaterialLocation::isOccupied(), and therefore on PortMaterialLocation, which inherits from it.

Alternatively this could mean "check to see if there is Material in any PositionalContainer at the port." If so, the comment should be restated to make that clear.

PortResource::transferOrder() and ProcessResource::transferOrder()

This operation is an unnecessarily complicated means of specifying the required access order. The given method requires a separate operation to find out what the MaterialLocations are, and assumes that they are all to be used. Why not simply provide the sequence of MaterialLocations in the order they are to be accessed?

And even that is not necessary if the sequence doesn't change dynamically. Since the (Port)MaterialLocations of a PortResource or ProcessResource can be accessed by the inherited `MaterialLocationSequence materialLocations()` from MachineResource, all that is wanted is the convention that for a PortResource the (returned) Sequence is significant and reflects the required order of access.

We would also suggest adding an operation:

```
boolean accessIsRandom()
```

which returns true if the MaterialLocations can be accessed in any order.

MaterialLocation::content

The attribute of Material called "format" corresponds to the attribute of a MaterialLocation or Container called "content." The latter is a poor choice since the true "content" of a MaterialLocation or Container is its "currentMaterial" or "containedMaterial."

Recommendation: rename MaterialLocation::content to "contentType" or "contentFormat."

MaterialLocation status model

For MaterialLocation, the model of "status" is badly confused by attempting to make it like "states" of other objects. In particular,

a. makeAllocated() allocates the MaterialLocation to receive a particular Material, and it could fail, but there is no exception, and there is no way to interrogate the MaterialLocation for the allocated Material. isAllocated returns boolean, rather than what the MaterialLocation is allocated to.

   Recommendation: change makeAllocated(in Material aMaterial) to:

```
allocateTo(in Material aMaterial)
    raises (MaterialLocationNotAvailable);

/* Return the Material to which this MaterialLocation is allocated,
 * if any, else NIL. */
Material allocatedTo();
```

   One can leave boolean isAllocated(), for clients who don't need the details.

b. isInService is a "state" that is independent of all the others. That is, a MaterialLocation can be simultaneously InService and Occupied. So the model is:

```
attribute boolean isInService;
```

There is nothing wrong with the current methods, except that they follow a pattern that is elsewhere only applied to mutually exclusive states.

c. makeOccupied() and makeNotOccupied() are totally inappropriate. The model is:

```
boolean isOccupied() { material() != NIL };
```

That is, isOccupied() is simply the interpretation of the fact that the material() operation does not return NIL, and the means of modification is materialReceived() and materialSent().

Recommendation: makeOccupied() and makeNotOccupied() should be deleted,

d. isNotOccupied is not a useful function. First, it is the same as ~ isOccupied(). (And there are no parallel isNotAllocated or isNotAvailable functions.) Second, isAvailable() is the question that the client *should* ask.

Recommendation: delete isNotOccupied().

e. The comment on isAvailable() says:

```
/* Available = InService and not Allocated, Reserved, or Occupied. */
```

but there is no operation that "reserves" a MaterialLocation. It is possible that there should be.

Recommendation: Either remove the word "Reserved" from the comment, OR add reserveFor() and isReserved().

MaterialLocation::resourceOwner

The comments say that a MaterialLocation "is a place in a MachineResource," and the definition of MachineResource confirms that. But method resourceOwner() is:

```
/* Return the owning Resource. */
Resource resourceOwner();
```

Recommendation: change it to

```
MachineResource resourceOwner();
```

Qualification of MaterialLocation "names"

The qualification of MaterialLocation "names" is correct, but inadequately described.

Recommendation:

a. Change the comment on machineName() from

```
/* Return the name of this MaterialLocation qualified to the Machine. */
```

to read:

```
/* Return the name of this MaterialLocation qualified to the Machine
 * that owns the MachineResource, i.e.\ the resourceOwner(). */
```

b. Change the comment on resourceName() from

```
/* Return the name of this MaterialLocation qualified to the
 * MachineResource. */
```

to read:

```
/* Return the name of this MaterialLocation qualified to the owning
 * MachineResource. */
```

MaterialLocation should not inherit from Resource

MaterialLocation is said to inherit from Resource. This implies that it must support startup(), shutdown(), etc. This is almost certainly wrong, and nothing else useful is inherited directly from Resource. The resourceOwner() (which should be a MachineResource) should support startup(), shutdown(), etc. and the naming conventions are already supported by specific methods on MaterialLocation.

Recommendation: Change MaterialLocation to delete the inheritance from Resource.

LocationPortal should be a kind of MaterialLocation

LocationPortal has *no* modelled properties, except a many-to-many relationship with MaterialLocation. It is the subject of several transfer operations. It inherits from Resource a name and an ownership relationship, although what might own it is not clear (a MachineResource? a Machine?). It also inherits from Resource startup and shutdown operations which are completely inappropriate.

It seems that a LocationPortal – the point at which a change of ownership occurs – must be a physical *place* which is able to hold a Material or a Container. And that would make it a MaterialLocation of yet another specialized type. Thus it seems that a LocationPortal should inherit most methods from MaterialLocation: it too can have content, format, inService, allocation, etc.

Recommendation:

a. make LocationPortal inherit from MaterialLocation

b. make a new interface, e.g. StorageLocation, to inherit from MaterialLocation and provide the methods (moved from MaterialLocation):

```
/* Answer the portals the MaterialLocation may be accessed by. */
LocationPortalSequence portals();

/* Add a LocationPortal to the MaterialLocations's collection of
 * access portals. */
LocationPortal addLocationPortal (in LocationPortal aLocationPortal);

/* Remove a LocationPortal from the MaterialLocations's collection of
 * access portals. */
LocationPortal removeLocationPortal (in LocationPortal aLocationPortal);

/* Answer the current portal the MaterialLocation has been using. */
LocationPortal portal();
```

(It will be seen immediately that these are cognate to the three methods supported by LocationPortal objects.)

How is a PortMaterialLocation related to a LocationPortal?

It is not clear how a PortMaterialLocation is related to a LocationPortal, but the definition of LocationPortal certainly fits a PortMaterialLocation. It seems in fact that these are two different models of the same objects. If there is a real difference, at least one of the definitions needs to make clear what it is. If the above model of LocationPortal is adopted, it seems that one of PortMaterialLocation and LocationPortal can be deleted, since they have identical models.

A guess is that people modelling storage resources (like an ASRS) model their input/output ports as LocationPortals and think of MaterialLocations as storage locations, while people modelling process resources model their input ports as PortMaterialLocations, which to them is just a special kind of place to hold material – a MaterialLocation. But the CIMF has to reflect an integrated conceptual model, in which the critical question is: do these objects have the same properties and operations? And the answer appears to be: yes.

Do PortResources have MaterialLocations that are *not* PortMaterialLocations?

The definition says

```
/* A PortMaterialLocation is a location owned by a PortResource, and only
   PortResources may own PortMaterialLocations. A PortMaterialLocation is a
   specialized MaterialLocation used to hold a container at a port. Its use
   with a PortResource distinguishes it from other specializations of
   MaterialLocation.
 */
```

It appears that the second sentence is the definition. It should appear first. It appears that the first and third sentences say exactly the same thing, but what they are trying to say is: Every MaterialLocation in a PortResource is a PortMaterialLocation, and every PortMaterialLocation is owned by a PortResource. Is this correct? (What is not clear from the wording is whether a PortResource can *also* have MaterialLocations that are *not* PortMaterialLocations.)

How do the PortResource operations addPortMaterialLocation and removePortMaterialLocation relate to the inherited (from MachineResource) operations addMaterialLocation and removeMaterialLocation? Why should a PortResource have both?

Recommendation: Delete operations addPortMaterialLocation and removePortMaterialLocation.

F Detailed Comments on CIMF Process Definitions

Global Type definitions (4.1.1)

```
typedef struct setting_struct {
    string settingName;      // the name of the setting
                             // (e.g., "Temperature")
    any settingValue ;       // the value for the setting
                             // (e.g., 125)
    unit units ;             // the units of the setting
```

```
                                        // (e.g., "degC"
    } setting ;
```

This use of "any" is needless and complicates the CORBA interface for both the client and the server. The datatype of "settingValue" should be "string." For those cases in which settingValue needs to be interpreted as an integer or floating-point number, the server knows what to expect and most processing languages provide "built-in" conversion routines of one form or another. By making it "any," all one does is substitute the ORB-provider's favorite conversion routine and complicate the interface into a discriminated union.

ProcessDefinitionManager

```
/* This signal is raised when attempt is made to remove a
 * ProcessDefinition when it contains a sequence of ProcessDefinitions. */
exception ProcessDefinitionContainsASequenceNoRemoveSignal
    { string processDefinitionName ; } ;
```

Why?

```
/* This signal is raised when a ProcessDefinition removal operation fails.*/
exception ProcessDefinitionNotFoundSignal
    { string processDefinitionName ; } ;

/* This signal is raised when a ProcessDefinition retrieval fails. */
exception ProcessDefinitionRetrievalFailedSignal
    { string processDefinitionName ; } ;
```

This is the strange view of exception returns again. They should both be NotFound. Find by name is not a "retrieval." All that is returned is the object-reference.

```
/* Creates an atomic level ProcessDefinition (no sequence of
   ProcessDefinitions), places it by name in the collection of
   ProcessDefinitions, and sets the pointer to its Document. */
ProcessDefinition createProcessDefinitionNamed_inDocument
   (in string processDefinitionName,
    in ProcessDefinitionSequence processDefinitions,
    in DocumentSpecification aDocumentSpecification)
     raises (ProcessDefinitionDuplicateSignal);
```

The second argument to this method contradicts the comment. By definition, the sequence must be empty. Recommendation: delete the second argument, or change the comment to explain what this really means.

```
/* Creates a ProcessDefinition (with a sequence of ProcessDefinitions),
   places it by name in the collection of ProcessDefinitions, and sets
   the pointer to its Document. */
ProcessDefinition createProcessDefinitionNamed_withSequence_inDocument
   (in string processDefinitionName,
    in ProcessDefinitionSequence processDefinitions,
    in DocumentSpecification aDocumentSpecification)
     raises (ProcessDefinitionDuplicateSignal);
```

Some part of 4.2.4.3 needs to make clear what the meaning of the ProcessDefinitionSequence parameter (and attribute of ProcessDefinition) means. Is the new ProcessDefinition to be interpreted as the concatenation of the ProcessDefinitions in the sequence? Or is it to be interpreted as the sequence of (what will become) ProcessFlowNodes by assigning one ProcessFlowNode to each un-

derlying ProcessDefinition? And what is the relationship to the Document object? i.e. how does it define the process and reference these "included" ProcessDefinitions?

```
/* Creates the ProcessFlowContext for a particular Product (workpiece)
 * from a ProcessFlow. */
ProcessFlowContext createProcessFlowContext_forProduct
    (in ProcessFlow aProcessFlow, in Product aProduct);
```

This seems to be an incorrect substitute for what is really wanted: assignment of a ProcessFlowContext for a particular Lot. Only in the degenerate case will a (possibly subdivided) Lot consist of a single workpiece. I assume that the thinking here is to capture rework and feed-forward process modifications, but rework often involves changing the routing and the routing unit is a Lot. Note that the Materials Management component allows for decomposition and recomposition of Lots. Using a per-Product routing model here defeats the whole purpose of that model.

Changing only the process parameters for a particular workpiece should be addressed by some other means.

Notes: ProcessFlowContext and ProcessFlow

Although the OMT diagram shows a relationship between these classes, and there is an operation on ProcessDefinitionManager that creates a ProcessFlowContext from a ProcessFlow, there is no operation on the ProcessFlowContext that returns the associated ProcessFlow.

This relationship is "internal" to the component (implementation). The manufacturing executive deals only with ProcessFlowContext:: beginNextProcessOperation and currentProcessOperation. And therefore ProcessFlow should be an internal object to the component that exposes no operations. (The external object is a ProcessDefinition, from which the Flow is created.)

But the ProcessFlow object is what is attached to the ProductSpecification. This also seems incorrect. If the ProductSpecification is supported by a Product Data Management (PDM) system, for example, what will be attached is the ProcessDefinition object that defines the ProcessFlow.

Recommendation:

a. delete the ProcessFlow object.

b. change

```
/* Creates the ProcessFlowContext for a particular Product (workpiece)
 * from a ProcessFlow. */
ProcessFlowContext createProcessFlowContext_forProduct
    (in ProcessFlow aProcessFlow, in Product aProduct);
```

to

```
/* Creates the ProcessFlowContext for a particular Product (workpiece)
 * using its ProductSpecification. */
ProcessFlowContext createProcessFlowContext_forProduct
    (in Product aProduct);
```

ProcessStep

```
/*  The ProcessStep class specifies what to do to a material. It is the
    application of the product's specification of processing for a
```

```
    particular product given the product's history. It is created from a
    single (atomic level) ProcessDefinition  and contains settings and a
    ProcessResource set and a ProcessCapability.
 */
```

While the settings and resource set and capability are attributes of this class, the ProcessDefinition is not. This is an error – the ProcessDefinition is what gives meaning to the ProcessStep object. Add:

```
    /* the definition of the operation to be performed */
    stepDefinition: ProcessDefinition;
```

ProcessDefinition

```
    ProcessDefinition addFirst (in ProcessDefinition aProcessDefinition)
    ProcessDefinition add_after (...)
    ProcessDefinition remove (...)
```

The naming of these methods does not conform to the v1.3 naming conventions.

```
    /* Set and get the ProcessOperationSpecification to use for the
     * ProcessDefinition. */
    attribute ProcessOperationSpecification
        referenceProcessOperationSpecification;
```

This relationship is not on the OMT diagram 4.28, and disagrees with the 4.2.4.3 description of the model.

On the OMT diagram, the relationship is to ProcessFlowNode. In this, the OMT diagram is wrong. Only "Operation" Nodes have associated ProcessDefinitions.

On the OMT diagram, this relationship is one-to-many. That is, several different ProcessOperationSpecification nodes may reference the same ProcessDefinition. In this, the OMT diagram is correct. The attribute should be

```
    attribute sequence<ProcessOperationSpecification> ...
```

In fact, this models a not clearly useful "inverse" relationship. The important relationship is that each ProcessOperationSpecification node points to its definition (the ProcessDefinition), and (technically) many may point to the same one.

```
    /* Adds a time estimate for this ProcessResource set used in the
     * process. This information is used for planning and scheduling. */
    void addTimeEstimate_for
        (in long timeUnits, in ProcessResourceSequence aProcessResourceSet);
```

Should not "long" above be Duration? (The great advantage of Duration is that it reconciles "timeUnits" from different process engineers.)

Note that one can addTimeEstimate_for but not removeTimeEstimate_for. This suggests that the intended implementation is that the time value is added *only* for those Resources in aProcessResourceSet which already happen to be in the hidden "eligibleProcessResources" list (maintained by addSettings_for, removeSettings_from). This suggests that there is either a missing method or a missing exception here.

```
    /* The ProcessCapability to use for the ProcessDefinition. */
    attribute ProcessCapability requiredProcessCapability;
```

Suggest replacing the comment with:

```
/* The ProcessCapability required for a ProcessResource to perform
 * the process defined by this ProcessDefinition.  Note -- an
 * impaired ProcessResource might lose some of its Capabilities,
 * so this is not redundant with the eligible ProcessResource set. */

/* Remove settings for a set of ProcessResources.  The sequence of settings
 * and ProcessResources that are passed into the service do not have any
 * implied order. */
void removeSettings_from (in settingSequence settings,
     in ProcessResourceSequence aProcessResourceSet);
```

Is it intended that removal of a setting be conditional on *both* the Name and the Value of the setting matching the one in the list? Or just the Name? If the intent is Both, then some kind of exception should certainly be possible. And if the intent is only Name, then the method seems to be misdefined, in that the first argument should be simply:

```
in stringSequence settingNames;

/* Return the settings for a given ProcessResource.  The order of the
 * returned sequence of settings is unspecified. */
settingSequence settingsFor (in ProcessResource aProcessResource);
```

This method should "fail" and return an exception if aProcessResource is not in the eligible ProcessResources list. Returning the empty set implies that the ProcessResource is there and has an empty list of settings.

Relationship of ProcessDefinition to ProcessResources

The relationship between the ProcessDefinition object and the ProcessResources which can perform it is strangely modelled.

First, *no* such relationship is depicted in the OMT diagram. Rather the OMT diagram shows the relationship to ProcessResource to belong to the ProcessOperationSpecification. In this, the OMT diagram is not quite right. The real relationship is between the underlying ProcessDefinition and the eligible ProcessResources, and the ProcessOperationSpecification derives its eligible resources from that relationship.

Second, the relationship between the ProcessDefinition and the eligible ProcessResources seems to involve a "hidden object," since that relationship itself has attributes – timeEstimate and (control) settings. This hidden object need not appear on the OMT diagram, but it MUST be explained in the text.

Third, there is apparently an unstated rule that it is only possible to add an eligible ProcessResource to the list by supplying the associated "settings." That is, the means of adding a ProcessResource to the list is called addSettings_for. This is not at all clear from the comment, but there is no method for attaching the eligible ProcessResource by itself.

This rule fails to understand the engineering reality. A process engineer can determine by various means what resources can perform an Operation long before he can perform the experiments to determine the proper settings, or the estimated time. So there should really be methods to set/add/remove eligible ProcessResource relationships, in addition to methods to attach the settings and timeEstimates to those relationships.

Fourth, it is not at all clear that removeSettings_from actually removes the resources themselves from the eligible list, but there is no other way of removing a ProcessResource from that list.

Fifth, there is no method that returns the list of all eligible ProcessResources.

Recommendation: Add an attribute:

```
/* The ProcessResources which could (normally) perform the process
 * specified by this ProcessDefinition.
 * The sequence of ProcessResources does not imply preference. */
attribute ProcessResourceSequence eligibleProcessResources;
```

and possibly two additional methods:

```
void addEligibleProcessResource (in ProcessResource aProcessResource);
void removeEligibleProcessResource (in ProcessResource aProcessResource);
```

ProcessFlow

```
This class represents the sequence of ProcessDefinitions that define
how to manufacture the product. The processFlow itself is made up of
ProcessFlowNodes (graph nodes) which comprise a general graph
structure. The primary entry points to the flow are determined by using
the entryPoints service. The ProcessFlow can then be traversed using the
supplied protocol below. Each node in the graph that is a
ProcessOperationSpecification will be used to create a ProcessOperation
which can be used by the processResource.
```

First, the intent is that the ProcessFlow is in the general case a directed graph, having at least parallel flows and apparently optional flows. Therefore it is not the "sequence" of ProcessDefinitions.

Second, it is not clear from any of the text what "entryPoints" actually returns. If the ProcessFlow *is* a linear sequence, then does entryPoints() return that sequence, i.e. the sequence of all ProcessFlowNodes? And if the graph is *not* simply a linear sequence, then what "sequence" does entryPoints() return, and what is the meaning of the ordering?

Finally, the last sentence is a comment on ProcessOpSpec and not on ProcessFlow.

```
/* Returns a sequence of ProcessFlowNodes that are the immediate
 * successors of this ProcessFlowNode. */
ProcessFlowNodeSequence nextProcessFlowNodesFrom
    (in ProcessFlowNode aProcessFlowNode)
    raises (ProcessFlowNodesRetrievalFailedSignal);
```

This is misidentified as an operation on a ProcessFlow. It is, by its nature, an operation on a ProcessFlowNode.

Also, if the "sequence" returned is all "immediate successors," then the "sequence" is logically a "set," that is, the ordering has no meaning. Or does it?

Recommendation: Either change "a sequence" to "the set," or specify the meaning of the ordering.

Question: Why is the interface set up to retrieve first/next *only for* ProcessOperationSpecifications? One would think that the scheme would apply to ProcessFlowNode generally. The implication of the interface as written is that intervening nodes that are not ProcessOperationSpecifications would be skipped. Is this intended?

ProcessFlowContext

```
/* Updates the current ProcessOperation to the next ProcessOperation in
```

```
     * the flow. This service is also used to start the ProcessFlowContext
     * "walk" by positioning at the first ProcessOperation. */
    void beginNextProcessOperation()
        raises (NoMoreProcessOperationsSignal);
```

The nomenclature is misleading. Following v1.3 conventions, we recommend:

```
informNextProcessOperationStarted()
```

ProcessOperationSpecification

```
long processingTimeFor_on
    (in long numberOfProducts,
     in ProcessResource aProcessResource);
```

The use of "long" here is inappropriate. The datatype should be Duration.

G Other Miscellaneous Comments on CIMF

Person::responsibilities

The description of the operation in IDL is different from the description in English in the comments:

```
/* Return the responsibilities (objects within a factory) which are
 * assigned to a Person's responsibility category. For example, return all
 * of the DocumentSpecifications in the category "specification management".
 */
personResponsibilitySequence responsibilities () ;
```

As defined the operation returns *all* responsibilities as a list of pairs (category, responsibility-object). This is probably a useful operation, but not what the comment describes. If the intention is to return all responsibilities *in a specific category*, as the comment indicates, then the operation should be:

```
anySequence responsibilities (in string responsibilityCategory);
```

Person::personResponsibility

The comment does not correctly describe the IDL data structure:

```
/* This type represents a dictionary of key/value pairs that represents
 * Personal responsibility associations. The key is
 * the "responsibility category name" while the value is the set of factory
 * objects associated with that category. */
typedef struct personResponsibilitystruct {
    string responsibilityCategory ;
    any responsibility ;
} personResponsibility;
typedef sequence<personResponsibility> personResponsibilitySequence;
```

"This type" refers to "personResponsibilitySequence" (the second typedef, not the first) and is correctly described as a "dictionary" of key/value pairs. But in each pair the value is *one* factory object ..., not "the set of factory objects ...."

History::at

What sort of "key" is envisioned for the "any" in operation "at?"

```
/* returns the HistoryEvent at the key */
HistoryEvent at(in any historyKey)
    raises (KeyNotFoundSignal);
```

The only attribute modelled for type HistoryEvent is a TimeStamp. Subtypes of histories that have subtypes of events with other keys will need to model the appropriate key search operations. Recommendation: Change "any" to "TimeStamp," viz.:

```
/* returns the HistoryEvent at/nearest the specified time key */
HistoryEvent at(in TimeStamp historyKey)
    raises (KeyNotFoundSignal);
```

History::from_to

```
/* returns a collection of all the HistoryEvents that occurred between
 * startingEvent to endingEvent (inclusive) from the History. */
HistoryEvent from_to
    (in HistoryEvent startingHistoryEvent,
     in HistoryEvent endingHistoryEvent)
    raises (KeyNotFoundSignal);
```

According to the comment this function should return type HistoryEventSequence.

ProcessRun::value

ProcessRun inherits attribute "value" from HistoryEventData, where it is declared type "any." What is really intended is:

```
/* Set and get the value of the HistoryEventData. */
attribute ProcessRunInformation value;
```

Since type "any" has a *very* high programming and support overhead in CORBA implementations, such "template" methods and attributes as HistoryEventData::value should be commented out (until IDL acquires "virtual" operation declarations) and the overrides for specific subtypes should be included in the running IDL.

ProcessRunInformation::inputs()

Are these the same as "settings?"

Since ProcessRunInformation is attached to the ProcessResource and not to the Product or ProductSpecification, it appears that either the concept "inputs" is heavily overloaded, or there is a missing attribute: the Lots processed during the run. Suggest adding:

```
/* Answer a sequence of the Lots processed during this run,
 * in the order they were processed. */
LotSequence lotsProcessed();

/* Add a Lot at the end of the sequence of Lots processed
 * during this run. */
```

```
void addLot(in Lot aLot);
```